(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 1 215 842 A2**

(12) **EUROPEAN PATENT APPLICATION**

AA

(43) Date of publication:
19.06.2002 Bulletin 2002/25

(51) Int Cl.7: **H04L 9/06**

(21) Application number: 01309323.2

(22) Date of filing: 02.11.2001

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 13.12.2000 US 255562 P
26.06.2001 US 892310

(71) Applicant: Broadcom Corporation
Irvine, California 92618 (US)

(72) Inventors:
• Qi, Zheng
Milpitas, California 95035 (US)
• Buer, Mark
Gilbert, Arizona 85296 (US)

(74) Representative: Browne, Robin Forsythe, Dr.
Urquhart-Dykes & Lord
Tower House
Merrion Way
Leeds LS2 8PA (GB)

(54)   **Methods and apparatus for implementing a cryptography engine**

(57)   Methods and apparatus are provided for implementing a cryptography engine for cryptography processing. A variety of techniques are described. A cryptography engine such as a DES engine can be decoupled from surrounding logic by using asynchronous buffers. Bit-sliced design can be implemented by moving expansion and permutation logic out of the timing critical data path. An XOR function can be decomposed into functions that can be implemented more efficiently. A two-level multiplexer can be used to preserve a clock cycle during cryptography processing. Key scheduling can be pipelined to allow efficient round key generation.
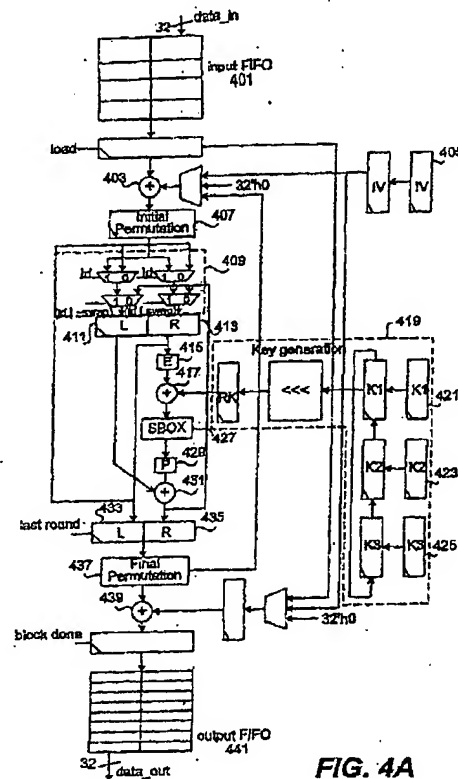
FIG. 4A

EP 1 215 842 A2

EP 1 215 842 A2

Description

Background of the Invention

5   1. Field of the Invention.

[0001]   The present invention relates to implementing a cryptography engine. More specifically, the present invention relates to methods and apparatus for efficient performance of a cryptography algorithm such as a DES algorithm.

10   2. Description of Related Art

[0002]   Conventional software and hardware designs for implementing various cryptography algorithms including the DES and the triple DES algorithms have significant performance limitations. Many designs for performing cryptography processing are well known in the art and are discussed, for example, in Applied Cryptography, Bruce Schneier, John
15   Wiley & Sons, Inc. (ISBN 0471128457), incorporated by reference in its entirety for all purposes. In order to improve the speed of cryptography processing, specialized cryptography accelerators have been developed that typically outperform similar software implementations. Examples of such cryptography accelerators include the BCM™ 5805 manufactured by Broadcom, Inc. of San Jose, CA.
[0003]   Although specialized hardware cryptography accelerators can often outperform software written to perform
20   the same tasks, conventional hardware cryptography accelerators have significant performance bottlenecks for implementing cryptography algorithms including the DES and the triple DES algorithms. Some performance bottlenecks include limitations related to key scheduling, clock rate, clock synchronization, and core logic operations.
[0004]   It is therefore desirable to provide methods and apparatus for improving the implementation of cryptography algorithms with respect to some or all of the performance limitations noted above.
25

Summary of the Invention

[0005]   Methods and apparatus are provided for implementing a cryptography engine for cryptography processing. A variety of techniques are described. A cryptography engine such as a DES engine can be decoupled from surrounding
30   logic by using asynchronous buffers. Bit-sliced design can be implemented by moving expansion and permutation logic out of the timing critical data path. An XOR function can be decomposed into functions that can be implemented more efficiently. A two-level multiplexer can be used to preserve a clock cycle during cryptography processing. Key scheduling can be pipelined to allow efficient round key generation.
[0006]   In one embodiment, a cryptography engine for performing cryptographic operations on a data block is pro-
35   vided. The cryptography engine includes a key scheduler configured to provide keys for cryptographic operations, multiplexer circuitry having an input stage and an output stage, expansion logic, and permutation logic. Expansion logic is coupled to the input stage of the multiplexer circuitry. The expansion logic is configured to expand a first bit sequence having a first size to an expanded first bit sequence having a second size greater than the first size, the first bit sequence corresponding to a portion of the data block. The output of the expansion logic is coupled to the input
40   stage of the multiplexer circuitry. The permutation logic is coupled to the expansion logic. The permutation logic is configured to alter a second bit sequence corresponding to the portion of the data block, whereby altering the second bit sequence performs cryptographic operations on the data block.
[0007]   The cryptography engine can also include an Sbox configured to alter a third bit sequence corresponding to the portion of the data block by compacting the size of the third bit sequence and altering the third bit sequence using
45   Sbox logic. In one embodiment, the cryptography engine is a DES engine. The cryptography engine can have two 2-to-1 multiplexers on the first level coupled to two 2-to-1 multiplexers on the second level. The two-level multiplexer can be configured to select either initial data, swapped data, or non-swapped data to provide to the output stage of the multiplexer.
[0008]   According to another embodiment, a cryptography engine for performing cryptographic operations on a data
50   block is provided. The cryptography engine includes a key scheduler configured to provide keys for cryptographic operations, multiplexer circuitry having an input stage and an output stage, expansion logic, permutation logic, and inverse permutation logic The expansion logic is coupled to the multiplexer circuitry. The expansion logic is configured to expand a first bit sequence having a first size to an expanded first bit sequence having a second size greater than the first size, the first bit sequence corresponding to a portion of the data block. The permutation logic is coupled to
55   the expansion logic. The permutation logic is configured to alter a second bit sequence corresponding to the portion of the data block, whereby altering the second bit sequence performs cryptographic operations on the data block. The inverse permutation logic is coupled to the input stage of the multiplexer circuitry. The inverse permutation logic performs the reverse operations of the permutation logic.

2

EP 1 215 842 A2

[0009] According to other embodiments, an integrated circuit layout associated with a cryptography engine for performing cryptographic operations on a data block is provided. The integrated circuit layout provides information for configuring the cryptography engine. The integrated circuit layout includes a key scheduler configured to provide keys for cryptographic operations, multiplexer circuitry having an input stage and an output stage, expansion logic, and permutation logic. The expansion logic is coupled to the input stage of the multiplexer circuitry. The expansion logic is configured to expand a first bit sequence having a first size to an expanded first bit sequence having a second size greater than the first size. The first bit sequence corresponding to a portion of the data block, wherein the output of the expansion logic is coupled to the input stage of the multiplexer circuitry. The permutation logic is coupled to the expansion logic. The permutation logic is configured to alter a second bit sequence corresponding to the portion of the data block, whereby altering the second bit sequence performs cryptographic operations on the data block.

[0010] According to other embodiments, an integrated circuit layout associated with a cryptography engine for performing cryptographic operations on a data block is provided. The integrated circuit layout provides information for configuring the cryptography engine. The integrated circuit layout includes a key scheduler configured to provide keys for cryptographic operations, multiplexer circuitry having an input stage and an output stage, expansion logic, permutation logic, and inverse permutation logic. The expansion logic is coupled to the multiplexer circuitry. The expansion logic is configured to expand a first bit sequence having a first size to an expanded first bit sequence having a second size greater than the first size, the first bit sequence corresponding to a portion of the data block. The permutation logic is coupled to the expansion logic. The permutation logic is configured to alter a second bit sequence corresponding to the portion of the data block, whereby altering the second bit sequence performs cryptographic operations on the data block. The inverse permutation logic is coupled to the input stage of the multiplexer circuitry. The inverse permutation logic performs the reverse operations of the permutation logic.

[0011] These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures, which illustrate by way of example the principles of the invention.

Brief Description of the Drawings

[0012] The invention may best be understood by reference to the following description taken in conjunction with the accompanying drawings.

Figure 1 is a diagrammatic representation of a system that can use the techniques of the present invention in accordance with one embodiment of the present invention.
Figure 2 is a diagrammatic representation of a cryptography engine having surrounding logic and a DES engine in accordance with one embodiment of the present invention.
Figure 3 is a diagrammatic representation of a packet that the DES engine may receive in accordance with one embodiment of the present invention.
Figure 4 is diagrammatic representation of a DES engine in accordance with one embodiment of the present invention.
Figure 5 is a diagrammatic representation of a DES engine that can be used for bit-sliced implementation in accordance with one embodiment of the present invention.
Figure 6 is a diagrammatic representation of a bit-sliced implementation that can be used in Figure 5 in accordance with one embodiment of the present invention.
Figure 7 is a diagrammatic representation of an optimized bit-sliced implementation in accordance with one embodiment of the present invention.

Detailed Description of Specific Embodiments

[0013] Reference will now be made in detail to some specific embodiments of the invention including the best modes contemplated by the inventors for carrying out the invention. Examples of these specific embodiments are illustrated in the accompanying drawings. While the invention is described in conjunction with these specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

[0014] Techniques are provided for enabling the implementation of efficient cryptographic processing, such as DES and triple DES processing. DES specifies encrypting individual 64-bit data blocks. A 64-bit data block of unencrypted

3

EP 1 215 842 A2

data is provided to the DES engine, combined with a key, and output as a 64-bit data block of encrypted data The key used for DES processing is typically a 56-bit number, although the key can be expressed as a 64-bit number. DES describes breaking up a 64-bit block of data into a right half and a left half, each 32-bits long. As will be appreciated by one of skill in the art, sixteen rounds of cryptographic operations are then performed. In each round, operations on the right half of the data include expansion, permutation, Sbox operations, and combination with a round key. A round key can be determined based on the round number of DES processing is referred to herein as a round key. The round key can be derived by applying permutation and shift functions to all 56 bits of the original key. The round number of DES determines the shift amount.

[0015] An XOR function is used to combine the right half of the data with a version of the key based on the round. The result can then be combined with the left half also by way of an XOR function. The unprocessed right half of the data becomes the left half of the data for the next round. Triple DES specifies performing three 16 round DES operations consecutively using three different keys. Typical hardware implementations for DES or triple DES perform one round calculation per clock cycle. The performance of DES or triple DES engine is therefore determined by the clock rate. However, a variety of hardware design constraints have traditionally limited the clock rate of cryptographic processing engines.

[0016] One design constraint that has traditionally limited the clock rate of cryptographic processing engines is the clock rate of circuitry surrounding the cryptographic processing unit. Surrounding logic having a slower clock rate than the DES engine is typically unable to provide data at a sufficient rate for DES processing. Methods and apparatus are described for decoupling the DES engine and the surrounding logic. The decoupled DES engine and surrounding logic allow flexibility in setting different clock ratios. The flexibility allows the DES engine to be designed to operate at a much higher clock rate than the surrounding logic.

[0017] Other design constraints relate to resistance capacitance (RC) delay. As will be appreciated by one of skill in the art, RC delay is a significant limitation in the performance of high-speed logic chips. RC delay in high speed circuit design can result from the resistance of interconnections and the capacitance associated with the dielectric media. RC delay is especially pronounced in sub 0.18 micron technologies. One way of limiting the effects of RC delay is to use bit-sliced design. That is, data paths are made narrower to allow more efficient custom-designs and layouts. In one example, the 32-bit data paths used in DES cryptography processing would be split into eight 4-bit data paths using efficiently designed layouts. Bit-sliced design, however, can not be used in conventional DES cryptography processing because the 32-bit data paths include expansion and permutation operations. The 32-bit blocks in the data paths are typically expanded, combined with a key, provided to an Sbox, and permuted. Bit-sliced design cannot be used because the bits in the 32-bit data blocks are intermingled. In other words, the bits in one four bit-slice would typically be swapped with bits from other four bit-slices because the bits are periodically permuted. Because there is no easy mechanism for keeping track of the bit swapping, bit-sliced design can not easily be implemented. Techniques are described for moving the permutation and expansion operations out of the 32-bit data path used in DES cryptography processing. By moving the permutation and expansion operations out of the data path, bit-sliced implementation can be used to limit the effects of RC delay in the time critical DES processing data path.

[0018] DES also describes the use of XOR operations in cryptography processing. As will be appreciated by one of skill in the art, XOR operations are computationally expensive. Techniques are provided for using more fundamental operations to replace XOR operations in the data path.

[0019] As noted above, the DES algorithm specifies sixteen rounds of processing. Triple DES performs the DES algorithm three consecutive times. During each round of DES processing, the left and right halves of the 64-bit data are swapped. However, in triple DES, the left and right halves of the 64-bit data are not swapped during the transitions between the consecutive performance of 16 round DES operations. To allow non-swapping of the left and right halves, logic that takes an extra clock cycle and a 3-to-1 multiplexer is typically used. Techniques using a two level 2-to-1 multiplexer are provided that do not take a extra clock cycle during the transitions between the sixteen rounds of DES processing. As will be appreciated by one of skill in the art, 2-to-1 multiplexers can be integrated together with registers.

[0020] The present invention also contemplates pipelined key scheduling to allow a round key to be provided to each DES round at a high clock rate. The key generation, selection, provision, and consumption operations can be separated into different stages for pipelined processing.

[0021] The techniques of the present invention can be implemented in a variety of contexts. Figure 1 describes the present invention in terms of a cryptographic accelerator system that can be implemented in a number of ways, such as for example, as a stand alone integrated circuit, as embedded software, or as a subsystem included in, for example, a server computer used in a variety of Internet and Internet related activities. It should be noted, however, that the invention is not limited to the described embodiments and can be used in any system where data encryption is desired.

[0022] Figure 1 is a diagrammatic representation of one example of a cryptographic processing system 100 in accordance with an embodiment of the invention. As shown in Figure 1, the present invention may be implemented in a stand-alone cryptography accelerator 102 or as part of the system 100. In the described embodiment, the cryptography accelerator 102 is connected to a bus 104 such as a PCI bus via a standard on-chip PCI interface. The processing

4

EP 1 215 842 A2

system 100 includes a processing unit 106 and a system memory unit 108. The processing unit 106 and the system memory unit 108 are coupled to the system bus 104 via a bridge and memory controller 110. Although the processing unit 106 may be the central processing unit or CPU of a system 100, it does not necessarily have to be the CPU. It can be one of a variety of processors. A LAN interface 114 couples the processing system 100 to a local area network
5   (LAN) and receives packets for processing and writes out processed packets to the LAN (not shown). Likewise, a Wide Area Network (WAN) interface 112 connects the processing system to a WAN (not shown) such as the Internet, and manages in-bound and out-bound packets, providing automatic security processing for IP packets.

[0023]   A cryptography accelerator 102 can perform many cryptography algorithms including the DES and the triple DES algorithms. It should be noted that although the techniques of the present invention will be described in the context
10 of DES and triple DES for clarity, one of skill in the art will appreciate that the techniques can also apply to other cryptography algorithms including variations to the DES and triple DES algorithms.

[0024]   Figure 2 is a diagrammatic representation of a cryptographic processing unit in accordance with one embodiment of the present invention. The cryptographic processing unit 201 has input and output buffers 203 and 205. The input and output buffers 203 and 205 may be coupled to an external processor as shown in Figure 1. The cryptographic
15 processing unit 201 contains a cryptography engine 207. The cryptographic processing unit 201 can also contain an authentication engine 217. As will be appreciated by one of skill in the art, other components including decoding logic may also be components in the cryptographic processing unit 201. The cryptography engine 207 contains surrounding logic 209 and DES engine 213. DES engine 213 can include asynchronous FIFO interfaces 211 and 215. The asynchronous FIFO interfaces 211 and 215 allow DES engine 203 to operate at a much higher clock rate than the surround-
20 ing logic 209. The asynchronous FIFO interface 211 also can be used to convert 32-bit blocks received from the surrounding logic 209 to 64-bit blocks for DES processing. The asynchronous FIFO interfaces 215 similarly he can be used to convert the 64-bit blocks for DES processing back to 32-bit blocks for the surrounding logic 209.

[0025]   The asynchronous FIFO interfaces 211 and 215 are provided to decouple the operations of the surrounding logic 209 in the DES engine 213. The surrounding logic 209 can parse the data to allow processing of portions of the
25 data by the DES engine 213. The DES engine 213 can process a data block whenever it is available at the input FIFO. The FIFO interfaces 211 and 215 compensate for the rate mismatch between the surrounding logic 209 and the DES engine 213. The asynchronous FIFO interfaces provide flexibility for setting different clock ratios between DES engine 213 and surrounding logic 209 and allow reuse of the same DES engine 213 design in other processor configurations. In one example, the clock ratio between the DES engine 213 and the surrounding logic 209 is 3 to 1. In this example,
30 the DES engine can operate at about 500 MHz while the surrounding logic can operate at about 166 MHz. The FIFO interface can be sized to provide sufficient data for the DES engine to process. The clock rate of the surrounding logic 209 as well as the latency to load and unload data for the DES engine 213 can be considered. In one example, the input FIFO 211 must be large enough to continually provide DES engine 213 with data. Each data block the surrounding logic processes can contain data that the DES engine 213 processes.

35 [0026]   While this embodiment of the invention is described with respect to a DES engine, the invention may be applied more generally to other cryptography engines such as AES.

[0027]   Figure 3 is a diagrammatic representation of data packets that the surrounding logic 209 may send to cryptography engine 207 by way of the FIFO interface 203 in accordance with one embodiment of the present invention. Packet 301 contains a header portion 303. Portion 305 contains keys and an initialization vector. Initialization vectors
40 (IV) are described in RFC 2405 titled The ESP DES-CBC Cipher Algorithm With Explicit IV, the entirety of which is incorporated by reference for all purposes. 'CBC' stands for cipher block chaining mode. This mode specifies that the cryptographic processing of 64-bit data blocks should be chained together in a way such that the result of last 64-bit data becomes the IV for processing the block follows. CBC mode can prevent a data block from being processed in parallel in many conventional systems.

45 [0028]   Portion 307 contains the payload. The payload is the portion of packet 301 that is provided to DES engine 213 for DES processing. The payload 307 can be followed by a trailer 309. The payload header 303, keys and initialization vector portion 305, and trailer 309 constitute the payload gap 311 associated with the data from surrounding logic 209. Data in the payload portion of the packet can be continually provided to the DES engine 213 by determining the payload gap 311 and the clock rate of the surrounding logic 209.

50 [0029]   As noted above, the DES engine 213 may be processing at 500 MHz while the surrounding logic 209 may only be able to provide data to DES engine 213 at a rate of about 166 MHz. The FIFO interface 211 can comprise registers also clocked at 166 MHz. Assuming that a single 32-bit word is bypassed every 166 MHz clock cycle, it would take 50 clock cycles at 166 MHz to bypass a payload gap 311 of 50 words. According to various embodiments, it takes 50 clock cycles to process a 64-bit block at 500 MHz. The 50 clock cycles at 500 MHz correspond to approximately
55 17 clock cycles at 166 MHz. The FIFO interface 211 would be configured to be large enough to hold at least four 32-bit blocks. That is, 64-bits of payload can be available in the FIFO interface 211 before DES engine 213 completes processing of a single 64-bit block. In one embodiment, the FIFO is sized to hold eight 32-bit blocks. Two 32-bit words are used to hold data between packets so that the DES engine knows the end of one payload. Therefore, six blocks are

5

EP 1 215 842 A2

available for payload. In some implementations, the payload gap can exceed 50 words. Gaps can include the following entries: MCW (1 32-bit word), Host Context (4 words), SA data (keys, 18 words), Buffer Descriptor (3 words), Ipv4 header and options (15 words), ESP header (IV, 4 words), ESP trailer (3 words), Status trailer (4 words), and a variable length Ethernet header and trailer.

[0030] Figure 4a is a diagrammatic representation of a DES engine 213 as shown in Figure 2. The DES engine can include an input FIFO 401. The input FIFO 401 can be asynchronous to decouple the DES engine from surrounding logic. A 64-bit data block is combined with an initialization vector from initialization vector block 405. Initialization vectors are described in RFC 2405 as noted above. The 64-bit block then undergoes an initial permutation at 407. The initial permutation occurs before round 1. It should be noted that in some variations to DES, to which the present invention is applicable, initial permutation and final permutation operations are not performed. The 64-bit block is then passed to two-level multiplexer stage 409. Two-level multiplexer stage 409 contains four multiplexers for determining whether to load initial data, swap data from the previous round, or not swap data from the previous round. Initial data is loaded in the first round of DES processing. Data is swapped between rounds of DES processing. Data is not swapped in triple DES between the completed 16 rounds of DES processing. Control logic (not shown) can track the round number in order to determine what signals to send to the multiplexers. The first level multiplexers are sent a signal to load or not to load data from a previous round or to allow initial data to pass through after initial permutation 407.

[0031] The second level multiplexers are provided a control signal instructing the multiplexers to swap or not swap the feedback data or load the initial data. The initial data is never swapped. The control signal can be the load value provided in the first level combined by a bitwise OR with the swap value in the right half and combined with the one's complement of the swap value in the left half. In a four-bit data path example, the load value is 1 and the swap value is 0. It should be noted that the control signals can be a variety of numbers including a sequence of ones or a sequences of zeros. In the case of loading initial data, initial data is loaded into the first level of multiplexers and provided to the second level of multiplexers. It can pass the second level of multiplexers if the load value is 1. According to various embodiments, the load value of 1 combined with a bitwise OR with the swap value of 0 is 1. The load value of 1 combined with a bitwise OR with the swap value of 1 is also 1. In the case of load feedback data, the load value is 0. The feedback data is swapped according to the value of signal swap. Registers 411 and 413 receive the initial data or the feedback data.

[0032] After the first round of processing, the load value is switched to 0. The swap value is now 1 to indicate that the left and right halves should be switched. Data is passed back along line 453 to the first level of multiplexers 409. Since the load value is 0, the data passed back along line 453 is selected and provided to the second level of multiplexers. The (ld|~swap) value for the multiplexer corresponding to register 411 is (0|0) so the right half passed back along line 451 is selected and provided to register 411. The (1d|swap) value for the multiplexer corresponding to register 413 is (0|1) so the data provided from the first level multiplexer corresponding to data from line 453 is sent to register 413.

[0033] One benefit of the two level multiplexers is that the timing critical 32-bit data is arranged to go through only the second level multiplexer, which can be combined with the registers 411 and 413 to minimize the delay.

[0034] As will be appreciated by one of skill the art, various control signals. An alternative approach would be to use a 3-to-1 single level multiplexer to select either the initial data, the swapped feedback data, or the non-swapped feedback data. The use of a 3-to-1 single level multiplexer however adds significant delay to the timing critical datapath because the 3-to-1 multiplexer can not be easily combined with the registers. The 3-to-1 multiplexer uses an extra clock cycle. According to a preferred embodiment, four 2-to-1 multiplexers are used. The 2-to-1 multiplexers in the second level of the multiplexer stage 409 can be integrated with the registers 411 and 413. Register 411 contains the last half of this initial 64-bit block in round 1. Register 413 contains the right half of the 64-bit block in round 1. Registers 411 and 413 both typically hold 32-bits of data. The 32-bit data block contained in register 413 is provided to both expansion stage 415 and to register 411 through multiplexer stage 409 for the next round. Control signals in multiplexer stage 409 are configured to provide the 32-bit data block contained in register 413 to register 411 in the next round of DES processing. The 32-bit data block is provided to expansion logic 415.

[0035] As will be appreciated by one of skill in the art, the expansion logic 415 changes the order of the bits in the 32-bit at block and also repeats certain bits. The expansion logic 415 uses the 32-bit block to generate a 48-bit block. The expansion logic improves the effectiveness of the encryption process and also makes the 32-bit block into a 48-bit block that corresponds to the size of the key. The 48-bit block can then be combined with an XOR with the 48-bit round key at 417.

[0036] Keys are provided by key generation logic or key scheduler circuitry 419. A version of the key for cryptography processing of the original 64-bit block is provided by key scheduler 419. Key scheduler 419 can provide a different version of the original key for every round by applying permutation and shift functions to all 56 bits of the original key. The 48-bit block resulting from the XOR at 417 is provided to Sbox stage 427. As will be appreciated by one of skill the art, each Sbox in Sbox stage 427 converts a six-bit input into a four-bit output. According to various embodiments eight Sboxes are provided in Sbox stage 427. Sboxes as well as other cryptography operations are described in Applied Cryptography, Bruce Schneier (ISBN 0471128457), the entirety of which is incorporated by reference for all purposes.

6

EP 1 215 842 A2

[0037]   The 32-bit output of Sbox stage 427 is provided to permutation stage 429. A permutation stage 429 maps input bits in certain positions to different output positions. In one example, bit 1 is moved to position 16 of the output, bit 2 is moved to position 7 of the output, and bit 3 is moved to position 20 of the output, etc. The 32-bit output of permutation stage 429 is combined with an XOR with the value in register 411 at 431. The result of the XOR is provided to the register 411 through multiplexer stage 409 for the next round of DES processing. The same process occurs for the subsequent rounds of DES processing. That is, the right half is expanded, combined with an XOR function with a version of the key, provided to an Sbox stage, permuted, and combined with an XOR with the left half. After the last round, the outputs are written to register 433 and register 435. The output can then undergo a final permutation at 437. The result of a final permutation at 437 is combined by way of an XOR with an initialization vector as noted above when the DES engine is used to decrypt data. Otherwise, the result of the final permutation at 437 can remain unchanged by combining by way of an XOR with a sequence of zeros.

[0038]   For triple DES, the outputs at 433 and 435 are passed back to multiplexer stage 409. Control circuitry determines how to pass the data back to register 411 and 413 for a next 16 rounds of DES processing. In typical implementations, an extra clock cycle is required to determine whether to swap the left and right halves or not to swap the left and right halves. According to the techniques of the present invention, however, two-level multiplexer stage 409 avoids the extra clock cycle. Data can be fed back to multiplexer stage 409 as if it were another round of DES processing. Control logic, allows the proper determination of how to load the data based on whether it is another DES round or a transition before another 16 round DES algorithm. More specifically, the swap value can be 0 for transitions. The processed data resulting from the DES or triple DES algorithm is provided to output FIFO 441. Output FIFO 441 can be asynchronous FIFO like input FIFO 401.

[0039]   Figure 4b is a diagrammatic representation of a DES engine in accordance with one embodiment of the present invention showing critical and non-critical paths in the round processing logic. A block of 64-bit data is input through multiplexer stage 409 into left register 411 and right register 413. The 32-bit data in right register 413 is provided to left register 411 for the next round of DES processing and to expansion block 415 as noted above in Figure 4a. The 48-bit output is combined using an XOR function with the round key stored in the key register 419 at 417. The 48-bit data block is then provided to Sbox stage 427 and permuted at 429. The logic operations performed such as XOR 417 and Sbox 427 on the right half data path are resource intensive and make path 451 a timing critical path. The timing critical path 451 can be implemented using bit-sliced design. Path 453 simply feeds the right half data back to the first level multiplexer Path 453 contains less logic than path 451 and is therefore referred to as a non-critical path.

[0040]   As noted above, bit-sliced implementation cannot be performed because the permutation 429 and expansion logic 415 change the order of bits in the 32-bit data block. It would be difficult to split a 32-bit data path into eight separate four bit slices or a 48-bit data path into eight separate six bit slices because the bits in the timing critical data path are intermingled by the expansion and permutation logic. The techniques of the present invention, however, contemplate moving the permutation and expansion logic operations out of the timing critical data path. By moving the expansion and permutation logic outside the timing critical data path, six bit slices can be custom-designed for combination with round keys and input into Sboxes. As noted above, Sboxes take six bit inputs and provide four bit outputs.

[0041]   The permutation and expansion logic are removed from the timing critical data path by recognizing the characteristics of the permutation logic. DES specifies that the permutation logic move bits from certain input positions to certain output positions. For example, the four bit sequence on 1, 0, 1, 1 can be permuted by moving bit 1 to bit 3, bit 2 to bit 4, bit 3 to bit 2, and bit 4 to bit 1. The resulting bit sequence would be 1, 1, 1, 0. The inverse permutation would move bit 1 to bit 4, bit 2 to bit 3, bit 3 to bit 1, and bit 4 to bit 2. The inverse permutation applied to the resulting bit sequence 1, 1, 1, 0 would yield the original bit sequence 1, 0, 1, 1.

[0042]   It can be recognized that taking the inverse permutation of the permutation of the bit sequence yields the original bit sequence. The relationship can be expressed as the following equations:

$$bitsequence = P^{-1}(P(bitsequence)) \qquad \text{Equation 1}$$

$$bitsequence = P(P^{-1}(bitsequence)) \qquad \text{Equation 2}$$

[0043]   Equations 1 and 2 can be applied to the DES engine shown in Figure 4. The L value at 463 is combined with an XOR the permutation of R value at 461 the yield T value at 465. The relationship can be shown with the following set of equations:

$$T = P(R) \, XOR \, L$$

7

EP 1 215 842 A2

$$T = P(R)\,XOR\,P(P^{-1}(L))$$

$$T = P(R\,XOR\,P^{-1}(L)) \hspace{4cm} \text{Equation 3}$$

[0044] From the equation, it can be recognized that the R value at 461 does not need to be permuted until after it is combined with an XOR with the modified L value. Since the R value does not need to be permuted, the permutation logic can be moved out of the timing critical data path.

[0045] Figure 5 is a diagrammatic representation of a DES engine in accordance with one embodiment of the present invention where the expansion and permutation logic are moved out of the timing critical data path. The T value at 515 corresponds to the T value at 465 in Figure 4. That is, $T = P(R\,XOR\,P^{-1}(L))$. More specifically, the L value at 513 is inverse permuted by inverse permutation logic 501. The R value at 511 is combined with an XOR with the inverse permutation of L value at 513. The result is then permuted by permutation logic 517 to yield T value at 515.

[0046] The data provided to the DES engine shown in Figure 5 is split into a right and left half The left half is provided to inverse permutation logic 501. The right half is provided to both inverse permutation logic 503 and expansion logic 505. The right half provided to inverse permutation logic 503 is passed to register 519 through two-level multiplexer 507. It should be noted that pulling the permutation and expansion logic out of the timing critical data path can be performed with or without using a two-level multiplexer.

[0047] The right half at register 519 is provided to register 521 for the next round of DES processing. The left half is provided to register 521 through two-level multiplexer 507. The left half or L value at 513 is combined with an XOR function with the R value at 511. The right half provided to expansion logic 505 is input into register 523 through two-level multiplexer 509. The value at 527 is then combined with an XOR with the round key and subsequently provided to Sbox stage 527. The result is R value at 511. The result is then fed back for the next round of DES processing. As noted above the two-level multiplexer determines whether to load the initial data, swap data, or not swap data. Since the permutation logic 517 and expansion logic 505 are both outside of the timing critical path, bit-sliced implementation can be used.

[0048] Figure 6 is a diagrammatic representation showing a bit-sliced implementation that can be used in accordance with one embodiment of the present invention. The value di_l is one of the inputs to first level multiplexer 601. The value di_rs is one of the inputs to first level multiplexer 603. The value di_r is one of the inputs to first level multiplexer 605. The value rd_slice_di is one of the inputs to second level multiplexer 611. The value di_r_mux is the output value of multiplexer 605 and the other input to multiplexer 611. The value 1d is the control signal provided to the multiplexers to determine what input should be provided to output. The 1d value can be a sequence of ones or a sequence of zeros. The value sel_out is the output of the multiplexer 611 provided to register 619. The value sel_out is combined with an XOR function with a round key at 621 to yield value sbox_di, According to a preferred embodiment, di_r_mux, rd_slice_di, sel_out, and sbox_di are all six-bit sequences. After sbox_di is provided to Sbox 625, a four-bit sequence sbox_do is output. Sbox_do is combined with an XOR with the four bit value from register 615 to output rd_slice_do.

[0049] Bit-sliced implementation is possible because the permutation and expansion logic has been moved out of the data path corresponding to rd_slice_di and di_r. Using a six bit slice implementation, circuitry can be custom-designed to limit the effects of RC delay in the data path corresponding to rd_slice_di and di_r.

[0050] As will be appreciated by one of skill in the art, RC delay is not the only factor limiting throughput. The XOR function is also typically slow. The data path can be further optimized by performing the XOR function together with the functions of the two multiplexers 605 and 611. Although XOR functions are typically slow, OR functions can be implemented more efficiently. The XOR function can be decomposed into a single OR function used with other logic operations. The decomposition can be accomplished by understanding the characteristics of the data at various logic operations. The sel_out value in register 619 is either di_r_mux or rd_slice_di. The determination is made by control signal sel equivalent to (ld|swap). That is, the di_r_mux value is selected if it is round one. The rd_slice_di value is selected if it is not round one. The swap value is based on whether the current data is in transition before a 16 round DES operation in triple DES. The selection can be represented by the following equations, where ~ is the one's complement, & is a bitwise AND, ^ is an XOR, and | is a bitwise OR. The value sbox_di can be represented by the following equations:

$$sel\_out = (set\ \&\ di\_r\_mux\ |\ \sim sel\ \&\ rd\_slice\_di) \hspace{2cm} \text{Equation 4}$$

[0051] Where sel is 111111, di_r_mux is selected as sel_out. Where sel is 000000, rd_slice_di is selected as sel_out, The value sel_out XORed with the key yields sbox_di, shown by the following equation:

8

EP 1 215 842 A2

$$\text{sbox\_di} = (\text{sel \& di\_r\_mux} \mid \sim\text{sel \& rd\_slice\_di})\ \hat{}\ \text{key} \qquad\qquad \text{Equation 5}$$

[0052] An XOR function (a ^ b) can be decomposed as (a & ~b | ~a & b). Equation 5 can be decomposed in the following manner:

$$\text{sbox\_di} = (\text{sel \& di\_r\_mux} \mid \sim\text{sel \& rd\_slice\_di})\ \&\ \sim\text{key} \mid$$

$$\sim(\text{sel \& di\_r\_mux} \mid \sim\text{sel \& rd\_slice\_di})\ \&\ \text{key} \qquad\qquad \text{Equation 5b}$$

$$\text{sbox\_di} = (\sim\text{sel \& rd\_slice\_di \&} \sim\text{key}) \mid (\text{sel \& di\_r\_mux \&} \sim\text{key}) \mid$$

$$\sim(\text{sel \& di\_r\_mux})\ \&\ \sim(\sim\text{sel \& rd\_slice\_di})\ \&\ \text{key} \qquad\qquad \text{Equation 5c}$$

$$\text{sbox\_di} = \text{rd\_slice\_di \& } (\sim\text{sel \& } \sim\text{key}) \mid$$

$$(\sim\text{sel \& rd\_slice\_di} \mid \sim(\sim\text{sel \& rd\_slice\_di})) \& (\text{sel \& di\_r\_mux \&} \sim\text{key}) \mid$$

$$(\text{sel} \mid \sim \text{id\_slice\_di}) \& (\sim\text{sel} \mid \sim \text{di\_r\_mux}) \& \text{key} \qquad\qquad \text{Equation 5d}$$

$$\text{sbox\_di} = \text{id\_slice\_di \& } (\sim\text{sel \& } \sim\text{key}) \mid$$

$$(\sim\text{sel \& rd\_slice\_di}) \& (\text{sel \& di\_r\_mux \&} \sim\text{key}) \mid$$

$$(\text{sel} \mid \sim\text{rd\_slice\_di}) \& (\text{sel \& di\_r\_mux \&} \sim\text{key}) \mid$$

$$(\text{sel} \mid \sim\text{rd\_slice\_di}) \& (\sim\text{sel} \mid \sim \text{di\_r\_mux}) \& \text{key} \qquad\qquad \text{Equation 5e}$$

$$\text{sbox\_di} = \text{rd\_slice\_di \& } (\sim\text{sel \& } \sim\text{key}) \mid$$

$$(\sim\text{sel \& rd\_slice\_di}) \& (\text{sel \& di\_r\_mux \&} \sim\text{key}) \mid$$

$$\sim\text{rd\_slice\_di \& } (\text{sel \& di\_r\_mux})\ \hat{}\ \text{key} \mid$$

$$\text{sel \& } (\text{sel \& di\_r\_mux})\ \hat{}\ \text{key} \qquad\qquad \text{Equation 5f}$$

[0053] The following term of Equation 5f is logically equivalent to 0: ~rd_slice_di & (sel & di_r_mux) ^ key | sel & (sel & di_r_mux) ^ key. Substituting in 0 yields Equation 6.

$$\text{sbox\_di} = \text{rd\_slice\_di \& } (\sim\text{sel \& } \sim \text{key}) \mid \sim\text{rd\_slice\_di \& } ((\text{sel \& di\_r\_mux})\ \hat{}$$

$$\text{key}) \mid \text{sel \& } ((\text{sel \& di\_r\_mux})\ \hat{}\ \text{key}) \qquad\qquad \text{Equation 6}$$

[0054] By substituting the following values, equation 7 can be derived.

$$\text{inp1} = \sim\text{sel \& } \sim\text{key}$$

$$\text{inp2} = ((\text{sel \& di\_r\_mux})\ \hat{}\ \text{key})$$

$$\text{inp3} = \text{sel \& } ((\text{sel \& di\_r\_mux})\ \hat{}\ \text{key})$$

9